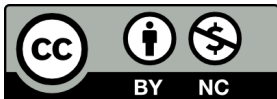


Business 4720 - Class 17

Recurrent Neural Networks using Python

Joerg Evermann

Faculty of Business Administration
Memorial University of Newfoundland
`jevermann@mun.ca`



Unless otherwise indicated, the copyright in this material is owned by Joerg Evermann. This material is licensed to you under the [Creative Commons by-attribution non-commercial license \(CC BY-NC 4.0\)](https://creativecommons.org/licenses/by-nc/4.0/)

What You Will Learn:

- ▶ Deep Learning Concepts
 - ▶ Recurrent Neural Networks
 - ▶ GRU cells and layers
 - ▶ LSTM cells and layers
 - ▶ Time series prediction with RNN
 - ▶ Business process prediction with RNN

Based On

Gareth James, Daniel Witten, Trevor Hastie and Robert Tibshirani: *An Introduction to Statistical Learning with Applications in R*. 2nd edition, corrected printing, June 2023. (ISLR2)

<https://www.statlearning.com>

Chapter 10

Kevin P. Murphy: *Probabilistic Machine Learning – An Introduction*. MIT Press 2022.

<https://probml.github.io/pml-book/book1.html>

Chapter 15

Tensorflow and Keras Tutorials

- ▶ https://www.tensorflow.org/tutorials/structured_data/time_series
- ▶ https://www.tensorflow.org/guide/keras/working_with_rnn
- ▶ https://www.tensorflow.org/text/tutorials/text_generation
- ▶ https://keras.io/examples/timeseries/timeseries_weather_forecasting/

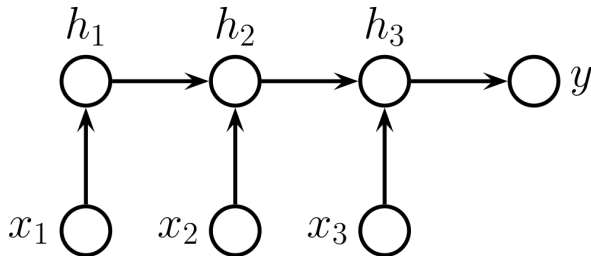
Other Tutorials

- ▶ <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- ▶ <https://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Application Examples

- ▶ Text classification
- ▶ Next-word or next-character prediction
- ▶ Text translation
- ▶ Time series forecasting (financial, ecological, meteorological, etc.)
- ▶ Business process prediction
- ▶ Speech translation or transcription
- ▶ Audio or sound generation
- ▶ Video captioning

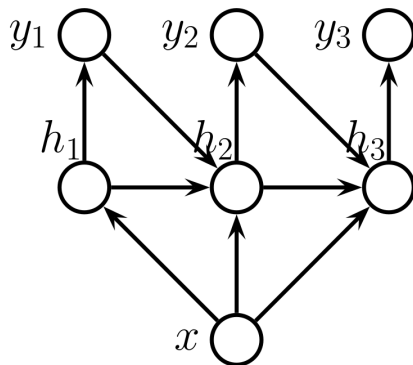
- ▶ Predict (regression or classification) from a sequence
- ▶ Inputs x , output y and hidden layers h



Source: Murphy Fig. 15.4

RNN – Vec2Seq

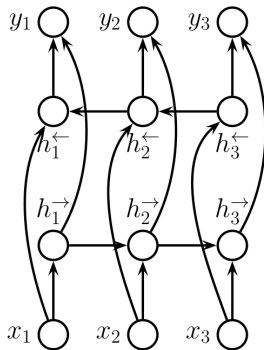
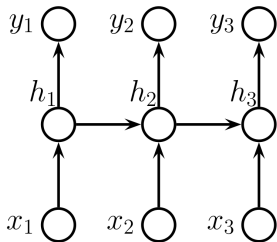
- ▶ Generate a sequence from initial input
- ▶ Input x , outputs y and hidden layers h



Source: Murphy Fig. 15.1

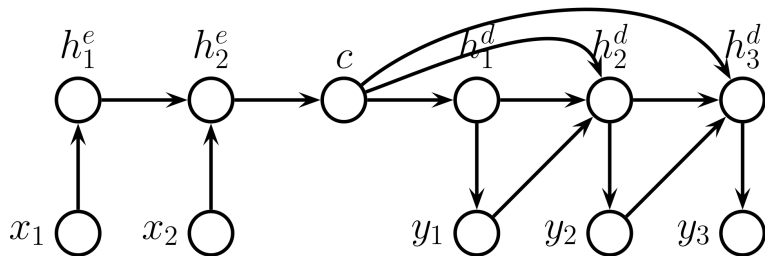
RNN – Seq2Seq

- Predict sequence from a sequence
- Inputs x , outputs y and hidden layers h



Source: Murphy Fig 15.5

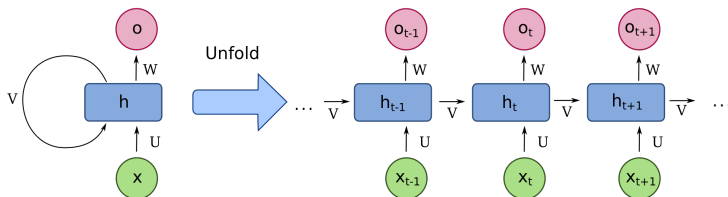
RNN – Seq2Seq (Non-Aligned)



Source: Murphy Fig. 15.7

Recurrent Neural Networks (RNN)

- ▶ A neural network block occurs multiple times in sequence
- ▶ **Input:** Sequence or single (initial) input
- ▶ **Output:** Sequence or single (final) output
- ▶ **State:** Hidden, passed from one step to the next



https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg

Backpropagation Through Time

$$h_t = \sigma(W_x \cdot x_t + W_h \cdot h_{t-1} + B_h)$$

$$o_t = \sigma(W_o \cdot h_t + B_o)$$

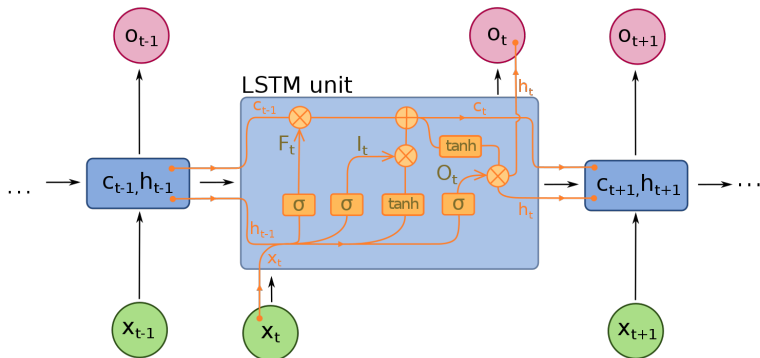
- ▶ Unfolding and truncating to make computationally tractable
- ▶ Train on short input subsequences

Vanishing Gradient Problem

- ▶ Multiplicative updates of state through time
- ▶ Loss of memory about inputs in distant past
- ▶ *Solution:* Additive updates of state

Long-Short-Term Memory (LSTM) Cells

- ▶ "Input gate" I , "Forget gate" F , "Output gate" O , candidate new memory \tilde{c}_t
- ▶ Hidden state h , Cell memory c



https://en.wikipedia.org/wiki/File:Long_Short-Term_Memory.svg

$$F_t = \sigma(W_f \cdot [x_t, h_{t-1}] + b_f)$$

$$I_t = \sigma(W_i \cdot [x_t, h_{t-1}] + b_i)$$

$$O_t = \sigma(W_o \cdot [x_t, h_{t-1}] + b_o)$$

$$\tilde{c}_t = \phi(W_c \cdot [x_t, h_{t-1}] + b_c)$$

$$c_t = F_t \otimes c_{t-1} + I_t \otimes \tilde{c}_t$$

$$h_t = O_t \otimes \phi(c_t)$$

Here \cdot is the dot-product (vector product), \otimes is element-wise multiplication, and $[.]$ denotes vector concatenation. σ is the sigmoid/logistic function and ϕ is the hyperbolic tangent.

LSTM Cell Parameters

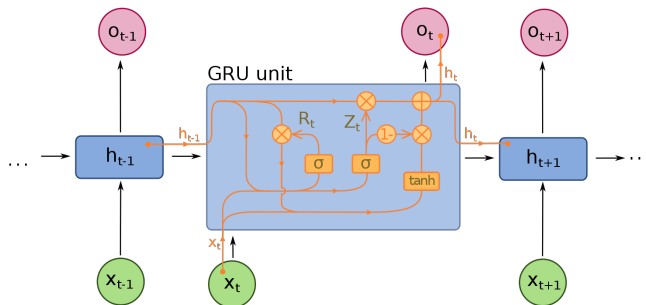
- ▶ Let h_t and c_t be vectors of size n and x_t be a vector of size m and
- ▶ The input to each gate is of size $n + m$, the output is of size n
- ▶ Then W_f , W_i , W_o and W_c must be matrices of size $n \times (n + m)$
- ▶ Then b_f , b_i , b_o and b_c must be vectors of size n .
- ▶ Then for each gate there are $n \times (n + m) + n$ parameters
- ▶ Then for all four gates, there are $4 \times (n \times (n + m) + n)$ parameters

Important: Parameters are "re-used" for each unrolled step.

Example: Let the unit size (state size) be 16 and the input size be 8.
Then the total number of parameters for the LSTM cell is
 $4 \times (16 \times (8 + 16) + 16) = 1600$

Gated Recurrent Unit (GRU) Cells

- "Update gate" Z_t , "Reset gate" R_t , candidate new memory \hat{h}_t
- Hidden state h



https://en.wikipedia.org/wiki/File:Gated_Recurrent_Unit.svg

- Fewer parameters than, and similar performance to LSTM

$$Z_t = \sigma(W_z \cdot [x_t, h_{t-1}] + b_z)$$

$$R_t = \sigma(W_r \cdot [x_t, h_{t-1}] + b_r)$$

$$\hat{h}_t = \phi(W_h \cdot [x_t, R_t \otimes h_{t-1}] + b_h)$$

$$h_t = (1 - Z_t) \otimes h_{t-1} + Z_t \otimes \hat{h}_t$$

Here \cdot is the dot-product (vector product), \otimes is element-wise multiplication, and $[.]$ denotes vector concatenation. σ is the sigmoid/logistic function and ϕ is the hyperbolic tangent.

GRU Cell Parameters

- ▶ Let h_t be a vector of size n and x_t be a vector of size m and
- ▶ The input to each gate is of size $n + m$, the output is of size n
- ▶ Then W_z , W_r , and W_h must be matrices of size $n \times (n + m)$
- ▶ Then b_z , b_r , and b_h must be vectors of size n .
- ▶ Then for each gate there are $n \times (n + m) + n$ parameters
- ▶ Then for all three gates, there are $3 \times (n \times (n + m) + n)$ parameters

Important: Parameters are "re-used" for each unrolled step.

Example: Let the unit size (state size) be 16 and the input size be 8. Then the total number of parameters for the GRU cell is

$$3 \times (16 \times (8 + 16) + 16) = 1200$$

With Keras GRU layer option `reset_after=False`.

Statefulness of RNN

Stateless LSTM/GRU

- ▶ Forget internal hidden state and cell memory after each training batch
- ▶ Allows shuffling of data between epochs
- ▶ No "long-term memory" for the LSTM/GRU

Stateful LSTM/GRU

- ▶ Retain internal state and cell memory between training batches
- ▶ Must not shuffle training data, data must be presented for learning in "correct" order
- ▶ Allows "long-term memory" for the LSTM/GRU across multiple batches

Hands-On Exercise

Consider a neural network for regression with the following characteristics:

- ▶ Layer 1: Embedding
 - ▶ Embedding matrix: 10000×10
- ▶ Layer 2: LSTM (seq2seq)
 - ▶ Hidden state and cell memory size: 24
 - ▶ Number of unrolled steps: 10
- ▶ Layer 3: GRU (seq2vec)
 - ▶ Hidden state size: 16
- ▶ Layer 4: Dense layer

What is the number of trainable parameters for each layer and for the whole network?

Stock Market Prediction

Using historic stock market data, predict future performance.

- ▶ DJIA performance
- ▶ "Seq2Vec" task: From a sequence of values, predict one following value
- ▶ Data exported from the R package `quarks`
- ▶ 2000 to 2021 (converted to EUR)
- ▶ Limited to open, low, high, close, and volume data

Load packages and read data file:

Python

```
import math
import tensorflow as tf
from tensorflow import keras
from keras import layers
import pandas as pd

tf.random.set_seed(123)
n_steps = 20
n_epochs = 25

data = \
pd.read_csv('https://evermann.ca/busi4720/djia.data.csv')
```

Add useful features for timeseries models:

- ▶ Successive differences
- ▶ Percentage changes

Python

```
data = pd.concat([  
    data,  
    data.diff().add_suffix('diff'),  
    data.pct_change().add_suffix('pct')],  
    axis=1).iloc[1:,]
```

Split data to train and validation set (no random shuffling for time series):

Python

```
train = data[:math.floor(0.8*data.shape[0])]
valid = data.drop(train.index)
```

Normalize data using only info from training set to prevent information 'leakage':

Python

```
train_mean = train.mean()
train_sd = train.std()
train = (train - train_mean)/train_sd
valid = (valid - train_mean)/train_sd
```

Create `tf.Dataset` objects for training and validation:

Python

```
dataset_train = keras.preprocessing \
    .timeseries_dataset_from_array(\
        train.drop('price.closediff', axis=1),\
        train['price.closediff'],\
        sequence_length=n_steps,\
        batch_size=32,\
        shuffle=True)
```

Python

```
dataset_valid = keras.preprocessing \
    .timeseries_dataset_from_array(\
        valid.drop('price.closediff', axis=1),\
        valid['price.closediff'],\
        sequence_length=n_steps,\
        batch_size=32,\
        shuffle=True)
```


Interlude – Dataset Objects

Dataset objects feed inputs and targets to the `fit` function.

See how this works with simple example data:

```
Python
test = pd.DataFrame([[0, 0], [1, 1], [2, 2], [3, 3],
                    [4, 4], [5, 5], [6, 6], [7, 7]])
inputs = test.iloc[:-1,0]
targets = test.iloc[2:,1]

pd.DataFrame([inputs, targets])
```

	0	1	2	3	4	5	6	7
0	0.0	1.0	2.0	3.0	4.0	5.0	6.0	NaN
1	NaN	NaN	2.0	3.0	4.0	5.0	6.0	7.0

Interlude – Dataset Objects

Do not shuffle data:

- ▶ Each batch continues from the previous batch
- ▶ Suitable for stateful RNN architectures

```
Python
ds = keras.preprocessing \
    .timeseries_dataset_from_array(
        inputs,
        targets,
        batch_size=2,
        sequence_length=2,
        sequence_stride=1,
        shuffle=False)

for element in ds.as_numpy_iterator():
    print(element)

(array([[0, 1],
       [1, 2]]), array([2, 3]))
(array([[2, 3],
       [3, 4]]), array([4, 5]))
(array([[4, 5],
       [5, 6]]), array([6, 7]))
```

Interlude – Dataset Objects

Shuffle data:

- ▶ Batches do not continue sequence
- ▶ Suitable for stateless NN architectures

Python

```
ds = keras.preprocessing \
    .timeseries_dataset_from_array(
        inputs,
        targets,
        batch_size=2,
        sequence_length=2,
        sequence_stride=1,
        shuffle=True)

for element in ds.as_numpy_iterator():
    print(element)

(array([[4, 5],
        [2, 3]]), array([6, 4]))
(array([[5, 6],
        [0, 1]]), array([7, 2]))
(array([[3, 4],
        [1, 2]]), array([5, 3]))
```

Sequence stride:

Python

```
ds = keras.preprocessing \
    .timeseries_dataset_from_array(
        inputs,
        targets,
        batch_size=1,
        sequence_length=2,
        sequence_stride=2,
        shuffle=False)

for element in ds.as_numpy_iterator():
    print(element)

(array([[0, 1]]), array([2]))
(array([[2, 3]]), array([4]))
(array([[4, 5]]), array([6]))
```

Hands-On Exercise – Dataset Objects

Using example data as in the previous slides,

- 1 Experiment with different values for `batch_size`,
- 2 Experiment with different values for `sequence_length`,
- 3 Experiment with different values for `sequence_stride`.

Do the results match your expectations?

Build a sequential model using an input layer, one LSTM layer and a dense (fully-connected) layer with a single output:

- ▶ Hidden state and cell memory size: `units=16`
- ▶ "Seq2Vec" model: `return_sequences=False`
- ▶ Stateless model: `stateful=False`

Python

```
model = keras.Sequential()
model.add(layers.InputLayer(
    input_shape=(n_steps, len(train.columns)-1)))
model.add(layers.LSTM(
    units=16,
    return_sequences=False,
    return_state=False,
    stateful=False))
model.add(layers.Dense(1))
model.summary()
```

Compile the model:

```
_____ Python _____  
model.compile(loss='mean_squared_error',  
              optimizer='Adagrad')
```

Fit the model to data:

```
_____ Python _____  
model.fit(dataset_train, epochs=n_epochs,  
          validation_data=dataset_valid)
```

Results:

```
Text
Model: "sequential_4"

Layer (type)                 Output Shape              Param #
=====
lstm_1 (LSTM)                 (None, 16)                1984
dense_8 (Dense)               (None, 1)                  17
=====
Total params: 2001 (7.82 KB)
Trainable params: 2001 (7.82 KB)
Non-trainable params: 0 (0.00 Byte)

Epoch 1/25
138/138 [=====] - 4s 14ms/step
loss: 1.1351 - val_loss: 7.2034

Epoch 25/25
138/138 [=====] - 1s 9ms/step
loss: 0.9916 - val_loss: 7.1444
```


Hands-On Exercises

Download the Python code from https://evermann.ca/busi4720/ts_prediction_s2v_stateless.py

- 1 Predict the percentage change of the closing value (column `price.closepct`)
- 2 Predict the actual closing value (column `price.close`)
- 3 Comment on the model performance results. Are these values more or less predictable than the differenced closing values?

Experiment with different model characteristics:

- 1 Vary the size of the LSTM state (`unit=16`).
- 2 Swap the LSTM layer for a GRU layer (`layers.GRU`). The GRU layer takes the same arguments as the LSTM layer.
- 3 Comment on the model performance results.

Can we do better with more layers?

Python

```
model = keras.Sequential()
model.add(layers.InputLayer(
    input_shape=(n_steps, len(train.columns)-1)))
model.add(layers.LSTM(
    units=16,
    return_sequences=True,
    stateful=False))
model.add(layers.Dropout(rate=0.25))
model.add(layers.LSTM(
    units=16,
    return_sequences=False,
    stateful=False))
model.add(layers.Dense(32))
model.add(layers.Dropout(rate=0.25))
model.add(layers.Dense(1))
model.summary()
```

The first LSTM returns sequences, the second one does not

Results of more complex model:

```
Text
Model: "sequential"

Layer (type)                 Output Shape              Param #
=====
lstm (LSTM)                   (None, 20, 16)           2048
dropout (Dropout)             (None, 20, 16)           0
lstm_1 (LSTM)                 (None, 16)               2112
dense (Dense)                 (None, 32)               544
dropout_1 (Dropout)           (None, 32)               0
dense_1 (Dense)               (None, 1)                33
=====
Total params: 4737 (18.50 KB)
Trainable params: 4737 (18.50 KB)
Non-trainable params: 0 (0.00 Byte)

Epoch 25/25
138/138 [=====] - 1s 8ms/step
- loss: 0.9966 - val_loss: 0.9486
```

Does a stateful LSTM perform better?

Set batch size to 1 for a single batch of continuous sequence:

Python

```
dataset_train = keras.preprocessing \
    .timeseries_dataset_from_array(
        train.drop('price.closediff', axis=1),
        train['price.closediff'],
        sequence_length=n_steps,
        sampling_rate=1, batch_size=1, shuffle=False)
```

Python

```
dataset_valid = keras.preprocessing \
    .timeseries_dataset_from_array(
        valid.drop('price.closediff', axis=1),
        valid['price.closediff'],
        sequence_length=n_steps,
        sampling_rate=1, batch_size=1, shuffle=False)
```

Stock Market Prediction [cont'd]

Python

```
model = keras.Sequential()
model.add(layers.InputLayer(
    batch_input_shape=(
        1, None, len(train.columns)-1)))
model.add(layers.LSTM(
    units=16,
    return_sequences=False,
    return_state=False,
    stateful=True))
model.add(layers.Dense(1))
model.summary()
```

Results of stateful model:

```
Text
Layer (type)      Output Shape      Param #
=====
lstm (LSTM)       (1, 16)           2048
dense (Dense)     (1, 1)            17
=====
Total params: 2065 (8.07 KB)
Trainable params: 2065 (8.07 KB)
Non-trainable params: 0 (0.00 Byte)

Epoch 1/25
4407/4407 [=====]
- 14s 3ms/step - loss: 1.0094 - val_loss: 1.4701

Epoch 25/25
4407/4407 [=====]
- 13s 3ms/step - loss: 0.9909 - val_loss: 1.1229
```

- 1 Download the stateful LSTM from `https://evermann.ca/busi4720/ts_prediction_s2v_stateful.py`
- 2 Extend the stateful model to a multi-layer network and include dropout layers
- 3 How does this change the predictive performance of the model?

How does a "Vec2Vec" model perform?

Treat the entire input as one large feature vector, without expressing any time-ordering in the model:

Python

```
model = keras.Sequential()
model.add(layers.InputLayer(
    input_shape=(n_steps, len(train.columns)-1)))
model.add(layers.Flatten())
model.add(layers.Dense(256))
model.add(layers.Dropout(rate=0.25))
model.add(layers.Dense(64))
model.add(layers.Dropout(rate=0.25))
model.add(layers.Dense(1))
model.summary()
```

The `Flatten` layer flattens its input tensor.

Results of "Vec2Vec" model. Note the large number of parameters.

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 300)	0
dense (Dense)	(None, 256)	77056
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 64)	16448
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 1)	65

=====
Total params: 93569 (365.50 KB)
Trainable params: 93569 (365.50 KB)
Non-trainable params: 0 (0.00 Byte)

Epoch 1/25 - loss: 3.5692 - val_loss: 2.3847
Epoch 25/25 - loss: 1.5864 - val_loss: 1.0902

Next Activity Prediction of a Business Process

Using an event log, train a model to predict the next activity from a prefix sequence of 5 activities that have already occurred.

Download the complete Python file here:

https://evermann.ca/busi4720/process_prediction.py

Download the example event log here: https://evermann.ca/busi4720/BPI_Challenge_2012.xes.gz

Python

```
import numpy
from tensorflow import keras
from keras import layers
import pandas as pd
import pm4py
# Length of sequences to predict from
prefix_len= 5
# Read the log
log = pm4py.read_xes('BPI_Challenge_2012.xes.gz')
```

Fix the data types:

Python

```
log['time:timestamp'] = \
    pd.to_datetime(log['time:timestamp'], utc=True)
log['case:REG_DATE'] = \
    pd.to_datetime(log['case:REG_DATE'], utc=True)
log['case:AMOUNT_REQ'] = \
    pd.to_numeric(log['case:AMOUNT_REQ'])
log['org:resource'] = \
    log['org:resource'].astype(str)
```

Filter the log for completion events:

Python

```
# Retain only activity completion events
log = log[log['lifecycle:transition'] == 'COMPLETE']
```

Filter and sort the log for sequences of length k :

Python

```
# Find the case start time as time of the
# first event in case
log = log.merge(
    log.groupby('case:concept:name', as_index=False) \
        ['time:timestamp'].min() \
        .rename(columns={'time:timestamp': 'case:start'}),
    how='left')
# Find the number of events for each case
log = log.merge(
    log.groupby('case:concept:name', as_index=False) \
        ['time:timestamp'].count(). \
        rename(columns={'time:timestamp': 'num_events'}),
    how='left')
# Filter log for minimum 6 events (5 input, 1 target)
log = log[log['num_events'] > prefix_len]
# Sort log by case start, then by event time
log.sort_values(['case:start', 'time:timestamp'], \
    inplace=True)
```

Identify feature to predict from:

```
_____ Python _____  
# This is the feature we predict (from)  
f_name = 'concept:name'  
# Vocabulary  
vocab = list(log[f_name].unique()) + ['EOC']  
v_size = len(vocab)  
# Dictionaries to convert to/from integers  
f2int = dict([(s, vocab.index(s)) for s in vocab])  
int2f = dict([(v, k) for (k, v) in f2int.items()])  
# Make sequences of feature names for each case  
features = log.groupby(['case:concept:name'])[f_name] \  
    .apply(list).reset_index(name='features')
```

Add end-of-case marker and convert to numeric:

Python

```
sequences=[l+['EOC'] for l in list(features['features'])]  
sequences=[[f2int[i] for i in seq] for seq in sequences]
```

Split sequences into prefix and target using a sliding window over each sequence:

Python

```
data = pd.DataFrame([(seq[i:i+prefix_len], \  
    seq[i+prefix_len]) for seq in sequences \  
    for i in range(len(seq)-prefix_len)])  
# Split the lists into dataframe columns  
data = data.assign(**data[0] \  
    .apply(pd.Series).add_prefix('index_'))
```

Create training and validation sets:

```
Python  
# Divide into train and test set  
train = data.sample(frac=0.8)  
valid = data.drop(train.index)  
# Separate X and Y  
train_x = train.iloc[:,2:]  
train_y = train.iloc[:,1]  
valid_x = valid.iloc[:,2:]  
valid_y = valid.iloc[:,1]
```

Define the model:

Python

```
model = keras.Sequential()
model.add(layers.InputLayer(input_shape=(5,)))
model.add(layers.Embedding(input_dim=v_size,
                           output_dim=16))
model.add(layers.LSTM(units=32,
                      return_sequences=False,
                      return_state=False,
                      stateful=False))
model.add(layers.Dense(v_size, activation='softmax'))
```

Note: `units` specifies the size of the h and c vectors

Compile and fit the model:

```
Python  
# Compile with loss and optimizer  
model.compile(loss='sparse_categorical_crossentropy',  
              optimizer='Adagrad',  
              metrics=['sparse_categorical_accuracy'])  
# Train the model and validate on  
model.fit(train_x, train_y,  
          validation_data=(valid_x, valid_y),  
          epochs=25, shuffle=True)
```

Results:

```
_____ Text _____
Layer (type)              Output Shape      Param #
=====
embedding (Embedding)     (None, 5, 16)     384
lstm (LSTM)                (None, 32)        6272
dense (Dense)              (None, 24)         792
=====
Total params: 7448 (29.09 KB)
Trainable params: 7448 (29.09 KB)

Epoch 25/25
2875/2875 [=====] - 6s 2ms/step
- loss: 1.3819 - sparse_categorical_accuracy: 0.6295
- val_loss: 1.3825 - val_sparse_categorical_accuracy: 0.6241
```

Predict activities from the model until end-of-case:

```
Python
input = train_x.iloc[2:3,:].copy()
print(input)
probs = model.predict(input)[0]
# pred = probs.argmax()
pred = numpy.random.choice(a=range(v_size), p=probs)
print(int2f[pred])
while int2f[pred] != 'EOC':
    for i in range(4):
        input.iat[0,i] = input.iat[0, i+1]
    input.iat[0,4] = pred
    print(input)
    probs = model.predict(input)[0]
    # pred = probs.argmax()
    pred = numpy.random.choice(a=range(v_size), p=probs)
    print(int2f[pred])
```

Results:

```
Text
index_0 index_1 index_2 index_3 index_4
109280      6      7      8     10      9
1/1 [=====] - 0s 197ms/step
W_Nabellen offertes
index_0 index_1 index_2 index_3 index_4
109280      7      8     10      9      9
1/1 [=====] - 0s 11ms/step
W_Nabellen offertes
index_0 index_1 index_2 index_3 index_4
109280      8     10      9      9      9
1/1 [=====] - 0s 11ms/step
W_Nabellen offertes
index_0 index_1 index_2 index_3 index_4
109280     10      9      9      9      9
1/1 [=====] - 0s 11ms/step
EOC
```

Hands-On Exercises

Download the complete Python file here:

https://evermann.ca/busi4720/process_prediction.py

Download the example event log here: https://evermann.ca/busi4720/BPI_Challenge_2012.xes.gz

Adapt the network architecture to identify the impact on training and validation performance of the following:

- 1 Dropouts in the LSTM layer (use the `dropout=0.x` option when defining the LSTM layer)
- 2 GRU instead of LSTM layers (use `layers.GRU()`)
- 3 Embedding size (originally 16, note: vocabulary size is 23+1)
- 4 Further training epochs (originally 25)

Comment on your findings and identify the best model.